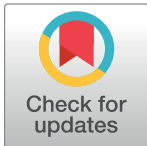


REVIEW

Better together: Elements of successful scientific software development in a distributed collaborative community

Julia Koehler Leman^{1,2}*, Brian D. Weitzner^{3,4,5,6}*, P. Douglas Renfrew¹, Steven M. Lewis^{7,8,9}, Rocco Moretti¹⁰, Andrew M. Watkins¹¹, Vikram Khipple Mulligan^{1,4,5}, Sergey Lyskov³, Jared Adolf-Bryfogle¹², Jason W. Labonte^{3,13}, Justyna Krys¹⁴, RosettaCommons Consortium¹, Christopher Bystroff¹⁵, William Schief¹², Dominik Gront¹⁴, Ora Schueler-Furman¹⁶, David Baker^{4,5}, Philip Bradley¹⁷, Roland Dunbrack¹⁸, Tanja Kortemme¹⁹, Andrew Leaver-Fay⁷, Charlie E. M. Strauss²⁰, Jens Meiler^{21,22,23,24}, Brian Kuhlman⁷, Jeffrey J. Gray^{3*}, Richard Bonneau^{1,2,25,26*}



1 Center for Computational Biology, Flatiron Institute, Simons Foundation, New York, NY, United States of America, **2** Dept of Biology, New York University, New York, NY, United States of America, **3** Dept of Chemical and Biomolecular Engineering, Johns Hopkins University, Baltimore, MD, United States of America, **4** Dept of Biochemistry, University of Washington, Seattle, WA, United States of America, **5** Institute for Protein Design, University of Washington, Seattle, WA, United States of America, **6** Lyell Immunopharma, Seattle, WA, United States of America, **7** Dept of Biochemistry and Biophysics, University of North Carolina at Chapel Hill, Chapel Hill, NC, United States of America, **8** Dept of Biochemistry, Duke University, Durham, NC, United States of America, **9** Cyrus Biotechnology, Seattle, WA United States of America, **10** Dept of Chemistry, Vanderbilt University, Nashville, TN, United States of America, **11** Dept of Biochemistry, Stanford University School of Medicine, Stanford CA, United States of America, **12** Dept of Immunology and Microbiology, The Scripps Research Institute, La Jolla, CA, United States of America, **13** Dept of Chemistry, Franklin & Marshall College, Lancaster, PA, United States of America, **14** Dept of Chemistry, University of Warsaw, Warsaw, Poland, **15** Dept of Biological Sciences, Rensselaer Polytechnic Institute, Troy, NY, United States of America, **16** Dept of Microbiology and Molecular Genetics, IMRIC, Ein Kerem Faculty of Medicine, Hebrew University of Jerusalem, Jerusalem, Israel, **17** Fred Hutchinson Cancer Research Center, Seattle, WA, United States of America, **18** Institute for Cancer Research, Fox Chase Cancer Center, Philadelphia PA, United States of America, **19** Dept of Bioengineering and Therapeutic Sciences, University of California San Francisco, CA, United States of America, **20** Bioscience Division, Los Alamos National Laboratory, Los Alamos, NM, United States of America, **21** Depts of Chemistry, Pharmacology and Biomedical Informatics, Vanderbilt University, Nashville, TN, United States of America, **22** Center for Structural Biology, Vanderbilt University, Nashville, TN, United States of America, **23** Institute for Chemical Biology, Vanderbilt University, Nashville, TN, United States of America, **24** Institute for Drug Discovery, Leipzig University, Leipzig, Germany, **25** Dept of Computer Science, New York University, New York, NY, United States of America, **26** Center for Data Science, New York University, New York, NY, United States of America

OPEN ACCESS

Citation: Koehler Leman J, Weitzner BD, Renfrew PD, Lewis SM, Moretti R, Watkins AM, et al. (2020) Better together: Elements of successful scientific software development in a distributed collaborative community. *PLoS Comput Biol* 16(5): e1007507. <https://doi.org/10.1371/journal.pcbi.1007507>

Editor: Dina Schneidman-Duhovny, Hebrew University of Jerusalem, ISRAEL

Published: May 4, 2020

Copyright: This is an open access article, free of all copyright, and may be freely reproduced, distributed, transmitted, modified, built upon, or otherwise used by anyone for any lawful purpose. The work is made available under the [Creative Commons CC0](https://creativecommons.org/licenses/by/4.0/) public domain dedication.

Funding: The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript. The U.S. National Institutes of Health software infrastructure grant R01-GM078221 supported refactoring, testing, documentation, and scientist training. We are also grateful for financial support from the Alfred P. Sloan Foundation, the Beckman Foundation, the European Research Council (Grant 310873), the Howard Hughes Medical Institute, the Israel Science Foundation (grant 2017717), the National Science Center in Poland (grant 2018/29/B/ST6/01989), the RosettaCommons, Los Alamos National Lab (grant LANL LDRD 20160044DR), the Packard Foundation, the Simons Foundation, USA-

* These authors contributed equally to this work.

¶ Membership of the RosettaCommons Consortium is listed in the [S1 List](#).

* julia.koehler.leman@gmail.com (JKL); bweitzner@lyell.com (BDW); jgray@jhu.edu (JJG); bonneau@nyu.edu (RB)

Abstract

Many scientific disciplines rely on computational methods for data analysis, model generation, and prediction. Implementing these methods is often accomplished by researchers with domain expertise but without formal training in software engineering or computer science. This arrangement has led to underappreciation of sustainability and maintainability of scientific software tools developed in academic environments. Some software tools have avoided this fate, including the scientific library Rosetta. We use this software and its community as a case study to show how modern software development can be accomplished

Israel Binational Science Foundation (grants 2009418, 2015207), the U.S. National Institutes of Health (F32-CA189246; P01-U19AI117905, R01-AI113867, R01-DK097376, R01-GM073151, R01-GM080403, R01-GM084453, R01-GM088277, R01-GM092802, R01-GM098101, R01-GM099827, R01-GM099842, R01-GM110089, R01-GM117189, R01-GM117968, R01-GM121487, R01-GM127578, R01-HL122010, R21-AI121799, R21-CA219847, R35-GM122517, RL1-CA133832, RL1-GM084433, T32-AI007244, T32-GM008403, U19-AI117905, UM1-AI100663), the U.S. National Science Foundation (CHE 1305874, CISE 1629811, DBI-1262182, DMR 1507736), and the Washington Research Foundation.

Competing interests: The Rosetta software has been licensed to numerous non-profit and for-profit organizations. Rosetta Licensing is managed by UW CoMotion, and royalty proceeds are managed by the RosettaCommons. Under institutional participation agreements between the University of Washington, acting on behalf of the RosettaCommons, their respective institutions may be entitled to a portion of revenue received on licensing Rosetta software including programs described here. Baker, Malmström, Yarov-Yarovoy, Gront, Meiler, Whitehead, Schueler-Furman, King, Gray, Sgourakis, Lindert, Strauss, Karanicolas, Bonneau, Sammond, Kortemme, and Bradley are unpaid board members of the RosettaCommons. As members of the Scientific Advisory Board of Cyrus Biotechnology, Baker and Gray are granted stock options. Yifan Song, Indigo C. King, Steven M. Lewis, Brandon Frenz, Karen Khar and Ryan Pavlovicz are currently employed at Cyrus Biotechnology with granted stock options. Cyrus Biotechnology distributes the Rosetta software. Brian D. Weitzner and Scott E. Boyken hold equity in Lyell Immunopharma. Vikram K. Mulligan is a co-founder and shareholder in Menten Biotechnology Labs, Inc. The content of this manuscript is relevant to work performed at Lyell and Menten. Neil P. King is a co-founder and shareholder of Icosavax, Inc., a biotech company developing nanoparticle vaccines. Justin B. Siegel is a co-founder and shareholder of Digestiva, Inc. and PVP Biologics Inc. David Baker is a co-founder, shareholder, or advisor to the following companies: ARZEDA, PVP Biologics, Cyrus Biotechnology, Cue Biopharma, Icosavax, Neoleukin Therapeutics, Lyell Immunotherapeutics, Sana Biotechnology, and A-Alpha Bio.

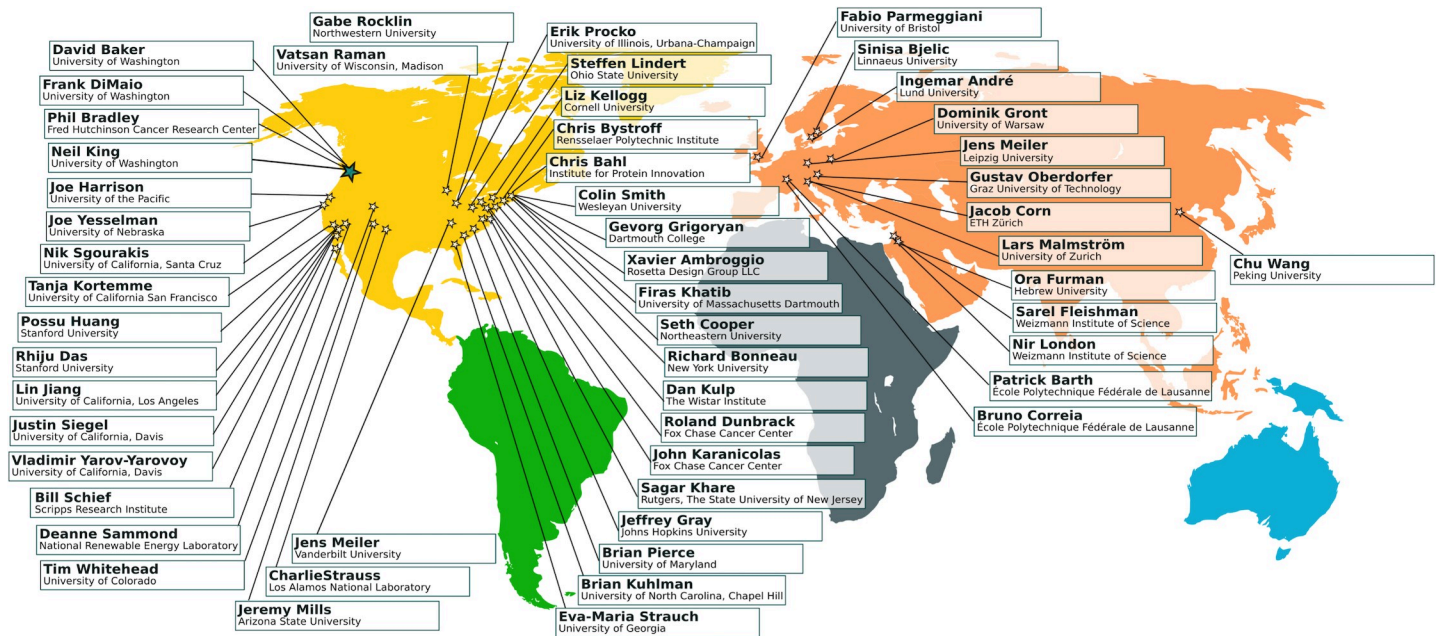
successfully, irrespective of subject area. Rosetta is one of the largest software suites for macromolecular modeling, with 3.1 million lines of code and many state-of-the-art applications. Since the mid 1990s, the software has been developed collaboratively by the RosettaCommons, a community of academics from over 60 institutions worldwide with diverse backgrounds including chemistry, biology, physiology, physics, engineering, mathematics, and computer science. Developing this software suite has provided us with more than two decades of experience in how to effectively develop advanced scientific software in a global community with hundreds of contributors. Here we illustrate the functioning of this development community by addressing technical aspects (like version control, testing, and maintenance), community-building strategies, diversity efforts, software dissemination, and user support. We demonstrate how modern computational research can thrive in a distributed collaborative community. The practices described here are independent of subject area and can be readily adopted by other software development communities.

This is a *PLOS Computational Biology* Software paper.

Introduction

Tackling grand challenges in various scientific disciplines requires ideas and perspectives from team members covering a diverse range of expertise, with sustained focus that persists over a time frame sufficiently long to meet bold scientific objectives[1,2]. But how do you assemble a team that can successfully work together on a shared goal over long periods of time? A recent National Academies report[1] summarized key challenges for “team science” as including diversity of membership, interdisciplinary knowledge integration, large size, goal alignment, geographic dispersion, and task interdependence[1]. We share here how our team has met these challenges. Our collective experience in the RosettaCommons consortium has been marked by extraordinary collaboration in a team composed of smaller groups that each innovate on their own projects. This teamwork is possible because our work centers around a shared software suite. The rise of shared, distributed scientific software tools across many scientific disciplines presents an opportunity for forming focused, productive research communities. In this paper, we share the key technical, social, and dissemination practices that have enabled successful scientific software development in our collaboration, the RosettaCommons [3]. The paper complements two recent, detailed, technical reviews about Rosetta that discuss latest applications[4] and the state and history of the score function[5].

Computational skills are increasingly vital across a broad range of scientific disciplines. However, few training programs in these fields include instruction in computer science[6] or software engineering. Thus, many scientists implement their workflows in transient scripts or programs focused only on addressing an immediate concern. These programs often lack generality, extensibility, code comments, or documentation, and can be difficult to integrate with other methods[7]. The general lack of documentation or example workflows (“protocol captures”[8,9]) further diminishes reproducibility and usability[10,11]. In contrast, accomplishing complex tasks and pushing scientific boundaries generally requires reuse of sophisticated, validated algorithms. Appropriately generalized, shared code can allow new, complex methods to be built from simpler, reusable modules that combine ideas from multiple sources. We have used such a model to develop the macromolecular modeling software suite Rosetta[4] (Fig 1). The project’s success and longevity have relied on the cultivation of a community around the



Rosetta software facts

Development started	1997
Language	C++
Founder	David Baker
RosettaCommons institutions (active)	71
RosettaCommons labs worldwide (PIs)	52
Developers	> 500 (~100 active)
Lines of code	3.1 million

Commercial licenses	80
Academic licenses	35,700 (Rosetta) 9,700 (PyRosetta)
Publications (estimate)	>1,500
Interfaces	PyRosetta (Python wrapper) RosettaScripts (XML scripts) Foldit (video game)
Recent success	KumaMax undergoing clinical trials for celiac disease

Fig 1. RosettaCommons facts. Main laboratories and institutions in the RosettaCommons and basic facts about our software in 2019.

<https://doi.org/10.1371/journal.pcbi.1007507.g001>

codebase with the larger goal of solving complex scientific problems by sharing ideas and tools and collaborating closely.

In addition to solving technical problems for scientific advancements, our community established coding, publication, and community-interaction standards that enabled continued growth and health of the software and our community over many years. We meet at least twice a year to discuss broad changes to the codebase and to share our newest tools and the scientific advances made with them. Additionally, we form new collaborations at these meetings and recognize members who have made important contributions to the codebase and/or the community. To ensure rapid entry into development, we organize regular training sessions for newcomers across the community. We created a code of conduct, which we require our members to adhere to, regardless of their institutions. We are closely connected to our extensive user base to further improve our software and apply it to real-world scientific problems. Our community has changed the way we work, especially in conjunction with relatively recent tools that allow social interaction on various levels (GitHub, Slack, video conferences).

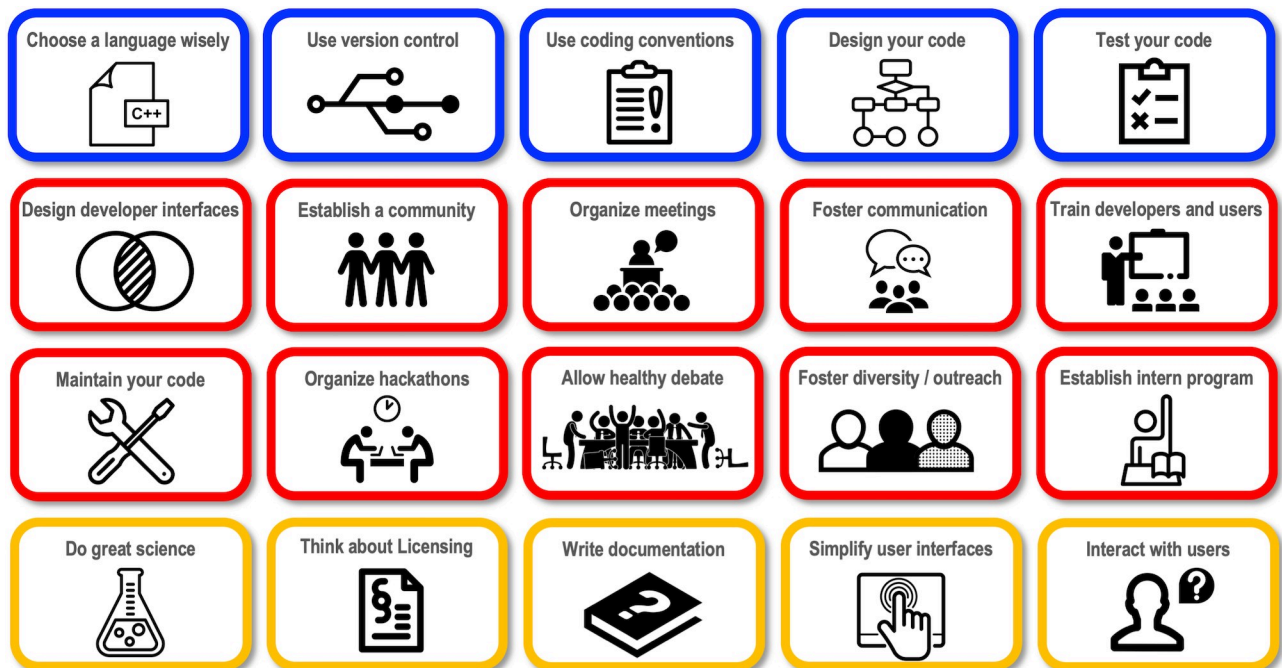


Fig 2. Pillars of success in the development of scientific software. Technical aspects are in blue, social aspects in red and dissemination and user interaction in yellow. This figure was created with icons from The Noun Project[13].

<https://doi.org/10.1371/journal.pcbi.1007507.g002>

Here we describe the social and technical elements that have shaped our software[4,12] and community over the past two decades, including mistakes and lessons learned. We focus on three major categories: technical aspects (*e.g.* version control and testing frameworks), social cohesion (*e.g.* meetings, communication, inclusion and outreach), and software dissemination (*e.g.* licensing, documentation, and user interaction). We believe that many of these features (Fig 2) can be translated to other communities. Thus, we hope this review can help other groups grapple with the many questions in performing collaborative research, from implementing effective software engineering practices to balancing healthy competition with effective use of shared resources, to inclusion of diverse groups of people and practicing transparent and inclusive decision-making strategies.

Beginnings and growth of our community

Development of our software suite started in the mid 1990s in David Baker's laboratory at the University of Washington (Fig 3). Rosetta was originally developed for protein structure prediction and to gain insights into protein folding, which remains a grand challenge in theoretical biophysics and underlies our understanding of biology, human health and disease. The project started when two graduate students implemented a Monte Carlo sampling algorithm [14] and a scoring function[15] that consisted of physics-based and knowledge-based terms. The protocol was successful in the blind prediction challenges, CASP 3[16] and 4[17]. Following this success, several postdoctoral fellows joined the project to develop various structure prediction and design applications.

As the early developers began independent positions at various institutions, they continued developing and improving their modeling applications, and it became clear that enhancing collaboration across institutions would be beneficial to scientific progress. This led to the creation of the RosettaCommons community of developers, which involved a unique intellectual

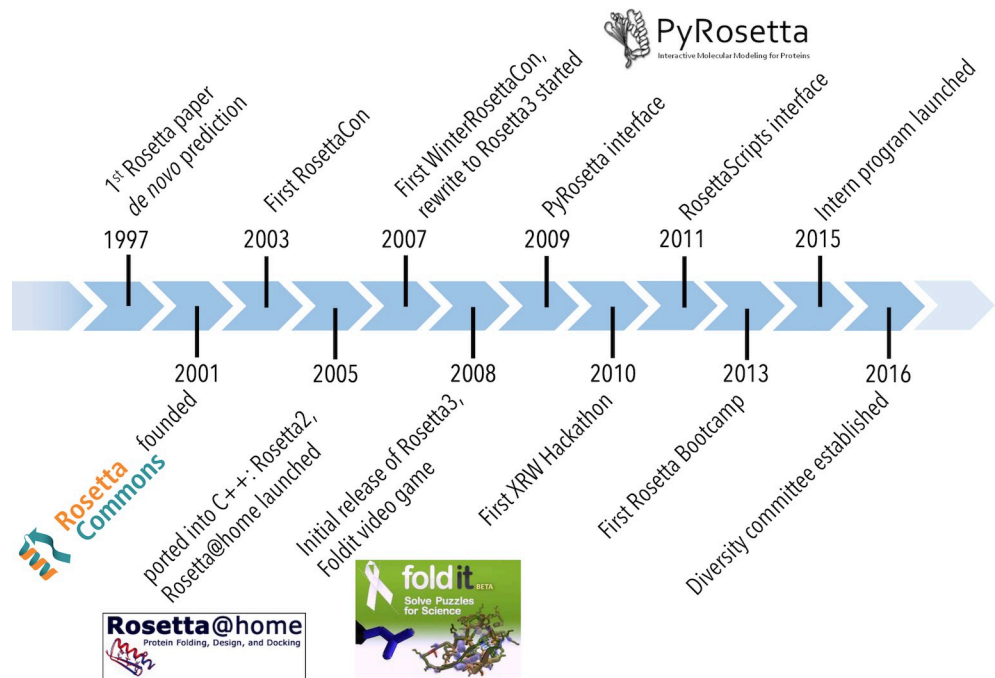


Fig 3. Rosetta history. Major milestones in the history of Rosetta development.

<https://doi.org/10.1371/journal.pcbi.1007507.g003>

property agreement, described in the licensing section below. The RosettaCommons structure allowed seamless collaboration across many institutions, which spurred developments in many different directions. By 2019, the RosettaCommons has grown to laboratories at 71 institutions worldwide, overseeing a project consisting of over 3 million lines of code with contributions from over 800 scientists. The code has been licensed to ~35,700 academic scientists and 80 commercial entities.

Technical aspects

Language history of the codebase

The early codebase was written in Fortran and later machine-translated into C++ (by Objexx Engineering), which was released in the summer of 2005 as Rosetta++ (also known as Rosetta2). Over the next two years, the automatic translation was found to be unworkable for continued development, and a software engineer was hired to implement the software as an object-oriented library, called *librosetta*. Although the first attempt at creating an object-oriented hierarchy turned out to be unwieldy and inflexible and ultimately had to be abandoned, it gave us insight into better choices for our central objects and their relationships. The next object-oriented rewrite of the codebase was initiated in the summer of 2007 by two of our core developers who were both scientists and software engineers (Andrew Leaver-Fay, a member of the Kuhlman lab, and Phil Bradley, at the Fred Hutchinson Cancer Research Center). This code was initially called *miniRosetta* and would later become Rosetta3, released in early 2009[12]. *MiniRosetta* also remedied a design flaw of Rosetta++ which presented as a single monolithic application with input options that allowed many different protocols to be invoked. In contrast, protocols in Rosetta3 are typically stand-alone applications with their own name. There are currently over 200 such applications.

Two scripting interfaces are also supported: *RosettaScripts*[18], which is an XML-based interface that allows protocol development from modular building blocks without additional

compilation, and *PyRosetta*[19,20], which allows Python-based protocol development and easy integration with third-party tools. The different interfaces are rooted in the Rosetta3 codebase from 2007, which is still used, developed and maintained at the time of this writing, 12 years later. In our case, after several iterations we found appropriate reconfigurable and extensible abstractions within an object-oriented programming framework in the C++ language to achieve our scientific objectives.

Version control

In software development, version control is important both to document changes to the software over time and to ensure that independent, parallel lines of development can be merged into a single, unified software. Version control is especially important for large codebases with complex dependencies of files and class hierarchies and where each revision in the main line of development needs to compile and run.

In the mid 1990s, our developers stored versions of the codebase in cryptically-named sub-directories. A single source code repository was created to which a single person had write permissions (Carol Rohl), using the then-popular Concurrent Versioning System (CVS[21]). CVS scaled poorly as the community of developers grew, and around 2006 the community moved to the centralized version control system Subversion (SVN[22]), which worked well for us for many years. As the number and geographic range of developers grew and the code complexity increased, our single SVN server became overwhelmed. Developers distant from the central server in Seattle encountered painfully slow or impossible merges due to timeouts, or had unsaved local changes erased. In 2013 we migrated to private repositories on GitHub[23], which uses the Git distributed version control system. Early on, we created conventions for interaction with GitHub (*e.g.* no forking, rewriting pushed commits, or force-pushing into the master branch). The decentralized nature of Git has served the community extremely well in supporting thousands of different development branches from all over the world that can be relatively easily merged into the main development branch. We have since instituted additional checkpoints for merges including automated testing (see below) and requiring independent code reviews from other developers in the community on each changeset (pull requests and reviews—see below). Pull request reviews have enabled us to improve code quality and reduce bugs. Administrative control of the code repository rests with senior developers, rather than principal investigators (PIs).

Testing framework

Code testing is necessary to ensure that software runs correctly and stably in different computing environments, provides expected and reproducible output, runs within a certain time-frame, and achieves some technical objective. Starting in 2004, we created our own testing framework because standard software packages at the time did not provide the features we needed. Testing has become an integral part of our software development. Because running tests locally on various architectures and build modes is challenging and time-consuming for individuals (see [S1 Text](#)), our framework includes a dedicated testing server on which tasks are distributed and results are collected ([Fig 4](#)). While establishing a custom test infrastructure might not be feasible for smaller software projects, several free and paid services are now available that are highly beneficial for scientific software (*e.g.* BuildBot[24], Jenkins[25], GitLab [26], TravisCI[27]). We support the following types of tests that are described in detail in the supplement: build tests, unit tests, integration (regression) tests, scientific tests, and others. RosettaCommons uses licensing revenue (see “Licensing” below) to acquire and maintain the hardware on which these tests are run.



Fig 4. Software testing. Testing is essential to ensure stability and correctness of software. This is a view of our testing server dashboard for a specific merge into the master branch of Rosetta. We continuously run a large number of tests, such as build tests, unit tests, integration tests, performance tests, code analysis tests, and scientific benchmarks. Tags in green, red, and yellow denote test pass, failure, and currently running.

<https://doi.org/10.1371/journal.pcbi.1007507.g004>

Improving code quality through code review

A requirement for collaboration and longevity of scientific software is a readable, maintainable, and extensible codebase as a foundation for new scientific development. In reality, however, everyone writes code differently. To create a base level of readability of the code, limit code breakage, and make code easier to debug and extend, we agreed on a set of about 200 coding conventions[28], to which everyone in the community is expected to adhere. Although it is challenging to strictly enforce all conventions, some are enforced by automated code quality tests, a custom “beautifier” script that formats code according to our conventions, and a Python-based code templating system for various common class types. Other conventions are socially enforced through code reviews (see below).

Several software development tools and frameworks have provided us with functionality that shaped our software development practices. We integrated features of Git and GitHub into our testing framework, including branching, pull requests, and code reviews (see S2 Text for workflow). A pull request is a request by a developer to merge new code from a branch into the main repository. The requestor can tag specific people to perform a review or broadcast a general call for reviewers. Based on the received comments, there can be multiple rounds of revisions. When all reviewers approve the merge, a final set of tests are run and the code is merged. This workflow is similar to software engineering industry practices and improves the quality of code contributions while also lowering the psychological barrier for new developers to contribute to the codebase without the worry of breaking it. In our experience, it can be challenging to maintain code quality, especially in large, diverse communities, and interactive code reviews have been a great benefit.

Interfaces to the main codebase

To facilitate participation by diverse new contributors, methods can be written in C++[12], PyRosetta[19] or RosettaScripts[18]. The choice for the interface is often made by considering technical details of the new method as well as the experience and comfort level of the developer. Protocols in RosettaScripts or PyRosetta (for which the C++ implementation is the common foundation) can be developed more quickly and require less technical knowledge but are subject to API changes and are frequently not merged back into the main codebase.

The first, simplest, and most-used interface to the codebase is the command line. Other interfaces were implemented over the years for different purposes and to facilitate custom protocol development. For example, in 2008, the **Foldit** video game[29,30] was released, which uses Rosetta as a foundation and employs a graphical user interface to allow the player to manipulate a protein structure. Major scientific concepts were adapted with a simplified terminology to make Foldit accessible to a broader audience. For instance, restraints are termed “rubber bands”, “shake” is used for rotamer optimization and design, and “wobble” for high-resolution refinement. The goal of the game is to achieve high scores rather than low energies. Foldit is widely used by the general public to solve real scientific puzzles[31], as a teaching tool for education levels from K-12 to graduate school[32], and by academic labs to gain insights and inspire new scientific discoveries.

In 2011 Fleishman *et al.* introduced **RosettaScripts**[18], which is an XML-based interface that allows custom protocol development by recombining and configuring existing objects. It takes advantage of the object-oriented framework of Rosetta3[12] and allows the user to create a new recipe without writing or compiling any C++ code. Intimate knowledge of the C++ codebase is not required, as the vast majority of the objects used in RosettaScripts are documented online[33]. Using RosettaScripts has been further facilitated by the XML Schema Definition (XSD) framework that identifies errors in the XML scripts at runtime, provides detailed error messages, and is now integrated into our user-facing documentation.

While RosettaScripts is useful for protocol development, it generally lacks flow control and detailed control of how objects can be manipulated. If these are needed and more specific control over certain objects are required for development, **PyRosetta** is the tool of choice. Since the early 2000's, the Python language has become the dominant programming language in many scientific disciplines, including biology and chemistry; largely because it is easy to learn, forgoes code compilation and features automatic memory and package management, while performing better than many other scripting languages. After initial work by William Sheffler, in 2009, Jeffrey Gray's group at Johns Hopkins University released PyRosetta[19,20], which enables the user to interact with nearly all underlying data structures *via* Python bindings (PyRosetta exposes 87,896 C++ functions and 7,403 C++ classes at the time of this writing). PyRosetta is for advanced users and beginning developers as it requires knowledge of the Rosetta libraries. A graduate course about protein structure modeling with PyRosetta is taught by Jeffrey Gray (videos are available at www.pyrosetta.org) and others in our community. Updated workshops are available on the website, and a book describing them was published in 2009[20].

Development in the C++ codebase is necessary for optimization of new protocols and the addition of new functionality and advanced features, such as new biomolecules or systems to model; examples include carbohydrates, nucleic acids, or symmetric systems.

Other interfaces include the CS-Rosetta toolbox[34,35] for incorporating NMR data from chemical shifts and residual dipolar couplings via Python wrapper tools, and InteractiveRosetta[36], which is a graphical user interface to PyRosetta[19].

Social aspects

The RosettaCommons community of developers—rewarding collaboration

The technical considerations outlined above reflect years of work and iteration by many scientific developers. As in many scientific software communities, the number of computer scientists and software engineers in our community is small compared to the number of scientists with domain expertise who have acquired programming skills without formal training. Developers go beyond applying existing code to current problems in that they also develop novel methods to address new scientific questions. By drawing from over 60 labs worldwide, the community includes developers with backgrounds in biochemistry, chemistry, physics, biology, computer science, mathematics, engineering, pharmacology, molecular physiology and other, related disciplines at the undergraduate, graduate, postdoctoral, and career professional levels (research scientists, principal investigators), and their collaborators. To fill these gaps in expertise (inevitable due to the large scope of the code) and work effectively together, we foster a culture that encourages communication, training, leadership, and outreach.

Achieving a robust codebase with scientific outcomes requires collaboration

Computer scientists and software engineers (especially in commercial settings) often emphasize correctness, code design, maintenance, readability, and performance. In contrast, academic scientists often favor rapid development of “messy” code to test a hypothesis, and scientists without formal computer science training emphasize scientific correctness and learn about software engineering concepts later or never. These factors lead to a codebase that is difficult to read, extend, maintain, and test, and which often performs poorly. In our community, we actively encourage, inspire, and teach these concepts to build a culture that appreciates and actively participates in code maintenance. To combat these barriers, sustainable software development requires (1) periodic code maintenance and (2) a culture of appreciation for software engineering concepts. We reinforce such a culture by giving out an annual *Rosetta Service Award*, which over the past ten years has honored contributions from code refactoring, hackathon leadership, diversity and inclusion efforts, and more. Anchoring these values into our community enables us to save a lot of time, effort, and money while interacting closely with community members allows us to implement new ideas more rapidly.

RosettaCommons membership

A central question is: who is the RosettaCommons? We seek to include all who share our scientific goals to develop Rosetta to predict structures and design macromolecules. In practice, the “community” intellectual property (IP), that is the code itself but not the products of the software, requires a legal sign-on from individual developers and their host institutions. The institutions and developers agree to contribute their intellectual property to the RosettaCommons codebase, to provide for an environment of continuous sharing and collaborative development for the advancement of public science. PIs are invited leaders of research groups, and developers in those groups sign a developer agreement. Historically, new PIs have been alumni from David Baker’s and other first-generation RosettaCommons labs. As our platform has been used in other areas of science (e.g. cryo-EM reconstruction or nanomaterials), new members have joined without training in specific Rosetta groups. The RosettaCommons executive board is currently evaluating possible strategies on how to expand the RosettaCommons to properly include the expanding set of developers.

Leadership: RosettaCommons Executive Board

The RosettaCommons was established in 2001[8] to facilitate collaboration between our laboratories at various institutions, and manage intellectual property and licensing of the codebase in collaboration with the University of Washington's technology transfer office, called UW CoMotion. Executive decisions are made by the principal investigators either at the *PI meeting* at the annual *RosettaCon* or via an email list. Since the RosettaCommons has grown substantially, we created an **executive board** in 2014 to oversee the community service work from communications to documentation, awards and budgets. PIs are elected to two-year terms for various board positions to match their expertise or interests, except for the conference chairs which rotate annually. The descriptions of the board positions can be found in [S3 Text](#).

Communication

As our software is developed by many labs worldwide, constant and efficient communication is crucial. Several **mailing lists** are in place for the C++ and PyRosetta developers, the PIs, test server logs to receive updates on code breakage, the diversity committee, and a jobs list, among others. We also have a **Slack** workspace for more interactive communication and for fast support within our community. We further have a **MediaWiki**[37] since the early 2000's communicating announcements for conferences and teleconferences. Most of the documentation has moved to a **Gollum wiki**[38], discussed under documentation below. Users can also ask questions on a searchable forum[39]. Monthly **TeleCons** allow the developers, the PIs, and the diversity committee to communicate. The variety of communication channels serve different purposes and further create a sense of community.

Community interaction in meetings and hackathons

The RosettaCommons spirit of close collaboration is fostered by regular meetings, the most significant of which is the annual **RosettaCon** in the summer. The inaugural meeting in 2003 was similar to a lab retreat, spending a few days in the North Cascades to share new developments. The last day of the meeting has traditionally been reserved for leisure activities as an opportunity to get to know the members of the community. Talking about science in a more relaxed setting has been a boon to our community spirit. For many people, **RosettaCon** feels more like friends solving scientific and technical problems together rather than a traditional scientific conference.

Summer **RosettaCon** has grown significantly to 270 people by 2017 ([Fig 5](#)). As the audience grew, the conference has shifted from mostly software discussion to also include scientific developments. All attendees present their research in a poster session or talk. The quality of the talks has increased substantially from less-formal talks (similar to group meetings) to well-polished presentations one would expect at international scientific conferences. Speakers range across all career levels.

In-depth code developments are discussed at **PreRosettaCon** (formerly called the *developer's meeting*) where major decisions about code maintenance, improvements, and restructuring are made. These discussions can be very technical and organizational. Senior developers typically organize the meeting, with decisions being made democratically by all attendees. We further organize an additional meeting, **WinterRosettaCon**, in February (formerly called *developer's meeting* or *MiniCon*, which started in 2007) with rotating locations. Similar to **PreRosettaCon**, these meetings have been smaller and more technical, covering both technical talks as well as discussion time for major decisions. In the last few years, however, **WinterRosettaCon** has grown to include a *bona fide* scientific program.



Fig 5. Meetings. Developers meet at our annual summer conference (RosettaCon). Pictures from our second RosettaCon in 2004 (left) and the conference in 2017 (right) reflect the community's drastic growth and increase in diversity.

<https://doi.org/10.1371/journal.pcbi.1007507.g005>

The community further organizes on-demand hackathons (*XRWs*—*eXtreme Rosetta Workshops*) for maintenance and improvement of the codebase. In these *XRWs*, about ten people from different research groups meet at one location to accomplish a specific, well-defined goal. Previous *XRWs* focused on improving compilation time and memory use by splitting the core libraries (2010) and the protocol libraries (2011); improving the scorefunction (Talaris meeting 2013 –this was technically not an *XRW* hackathon but a one-off technical meeting with presentations and discussions); improving documentation, protocol captures, and tutorials (2015 and 2016 –[Fig 6](#)); generalizing storage of chemical information and handling of common molecular file formats (2016); improving the handling of RosettaScripts input (2016); and generalizing interfaces for handling non-canonical amino acids, carbohydrates, nucleic acids, and metal-binding sites (2017). *XRWs* have had a tremendous impact on maintainability, extensibility, and user-friendliness of our codebase while further enhancing community spirit between participants.



Fig 6. Hackathons. Organizing regular hackathons can drastically improve code maintainability, generalizability, documentation, and interaction in the community. Our eXtreme Rosetta Workshops (*XRWs*) are organized annually and have had a drastic positive impact on our software and community.

<https://doi.org/10.1371/journal.pcbi.1007507.g006>

Training for users and developers

Rosetta is written with the goal of understanding complicated molecular systems at the interface between chemistry, physics, and biology. Since the software has primarily been developed by scientists rather than software engineers, it has historically been difficult to use. The creation of various user and developer interfaces (command line, PyRosetta, RosettaScripts) is a result of trying to address some of these challenges. In addition, our community has established a number of workshops to make novices familiar with the use of the software suite and its codebase as quickly as possible.

Users with a general understanding of protein science seeking an introduction to commonly-used protocols can follow a set of **tutorials** compiled by the lab of Jens Meiler and collaborating developers at Vanderbilt University; they supplement a manuscript describing the main protocols[9]. These tutorials are taught twice a year at Vanderbilt[40], and are probably the quickest way to learn how to use the software.

In 2013, we first began offering a week-long *Bootcamp* course (now called Code School) for small groups of interested graduate students and postdocs in RosettaCommons laboratories. The course covers intricacies of C++ programming and the architecture of our software libraries in lectures and in-class labs. We keep a high ratio of teachers to students (~1:5) so that multiple teachers can answer questions simultaneously during the lab sessions. The course was intended to train non-computer scientists to become senior developers and hopefully address our gender imbalance (more below). Videos of code school lessons are available on the RosettaCommons YouTube channel[41]. In 2019, we added a *PyRosetta Code School* course, which has an even lower barrier to entry as Python is an easier language to work with. *Code Schools* directly taught by senior developers have drastically improved code quality, consistency, readability, documentation, and understanding of important software engineering concepts. They have further improved the interaction and communication within the community and helped newcomers feel accepted.

Because gaining proficiency in software development typically takes many months to years of training, some members move between RosettaCommons laboratories throughout their career to apply their skills to different types of macromolecular modeling and design problems. These individuals, often having 6+ years of experience and institutional memory, play a crucial role in advancing and supporting the codebase, our community and training newcomers. Further, the RosettaCommons supports two research professors (Andrew Leaver-Fay and Rocco Moretti) who use their expertise to answer difficult technical questions and who are qualified to make major decisions about the codebase, and an experienced senior test engineer (Sergey Lyskov) who created and maintains the test infrastructure. Collectively, the 15–20 senior developers, scattered across multiple labs, have comprehensive knowledge of the ever-growing codebase and functionality of the software suite.

Diversity, Inclusion, and Outreach

Building and maintaining a strong community of collaborators requires fostering and recruiting top talent. As our community grew, we noticed that our team lacked gender and racial/ethnic diversity and that we were missing a sizeable fraction of the talent available. We believe an inclusive culture and a diverse membership are prerequisites to forming an ethical and just community of scientists, so we focused on recruiting and developing the scientists we failed to include previously.

In addition to offering *Code Schools*, two important first steps were instituting a Code of Conduct[42] and forging a diversity statement[43] to capture our values and aspirations. The Code of Conduct includes a reporting system that allows us to refer legal issues to home

institutions and to address microaggressions at conferences on-site. The diversity statement[43] commits to making “our community inclusive and equitable for people of all backgrounds, regardless of race, ethnicity, nationality, gender identity, sexual orientation, disability status, age, belief system, or socioeconomic background” (see S4 Text). It also “urges the individual research groups in our community to make every effort to proactively and swiftly eliminate discriminatory policies and procedures in hiring and recruitment and to make efforts to increase the number of individuals from historically underrepresented groups in the Rosetta community”.

As specific actionable items, RosettaCommons members have participated in a variety of outreach activities that fulfill several roles: (1) educating the general public about science, (2) sparking interest in STEM in K-12 and undergraduate students, (3) increasing diversity and inclusivity in the RosettaCommons, and (4) crowdsourcing scientific discoveries.

Our recruitment efforts include regular attendance at conferences that aim to highlight diversity in STEM fields such as the *Grace Hopper Celebration of Women in Computing*, the *Annual Biomedical Research Conference for Minority Students* (ABRCMS), the *ACM Richard Tapia Celebration of Diversity in Computing*, and the *Society for Advancement of Chicanos/Hispanics and Native Americans in Science* (SACNAS) conference. These conferences directly strengthen our climate and culture, as solo-status[44] Rosetta scientists in their home lab meet in person with those sharing their identities in other RosettaCommons labs and beyond, forging ties and networks that endure past these meetings. Professional development opportunities at these conferences empower scientists from minority groups and enlighten delegates from majority groups.

For undergraduates interested in a research experience in a RosettaCommons lab, we have established an NSF-funded REU (Research Experience for Undergraduates) program, the *Rosetta Summer Intern Program*[45]. Our program starts with a week-long *Intern Code School* where students learn how to use the software, how to navigate the code, and the most important frameworks[46]. After that week, students are able to develop basic protocols and accomplish specific research goals during their 8-week internship, results of which they present at RosettaCon.

Additional activities to improve inclusion in our community have included: (1) creating an active Diversity and Inclusion Committee (see Rosetta consortium supplement for members); (2) data collection on conference demographics; (3) invited platform speakers at RosettaCon to discuss diversity topics (Karen Fleming on gender equity, John Matsui on diversity 101, Joyce Yen on leadership and measuring inclusion); (4) networking/mentoring events at RosettaCon for women and members of minority groups; (5) RosettaCommons climate survey and follow-up small group discussions at RosettaCon resulting in recommendations and new actionable items; (6) dependent-care grants to support parents to attend Rosetta meetings; (7) conference housing matchups considering non-binary gender; (8) collecting resources for career development and personal well-being.

Our efforts have substantially increased the number of women in a typically male dominated field. Nearly 30% of *RosettaCon* attendees in 2017 identified as female, up from below 15% in most years before 2012 (S1 Fig). Almost 8% of attendees identify as Black, Hispanic, or multiracial. The demographics of our core developers and PIs are much less diverse, giving us humility and recognition that diversity efforts need to be sustained for many years to come. We will continue our work to ensure the RosettaCommons is a universally welcoming and supportive environment.

Public participation in science

In the early 2000s when protein structure prediction was much less tractable and compute resources were limited, we created a distributed computing project called **Rosetta@home**,

through which volunteers can donate idle computer cycles for protein structure prediction, docking, and design[47,48]. Rosetta@home was initially released in 2005 and runs on the Berkeley Open Infrastructure for Network Computing (BOINC)[49]. Rosetta@home shows a screensaver while performing computations when the user's desktop or laptop is otherwise idle. In 2015, Rosetta@home was ported to the Android operating system to run on volunteers' cell phones and mobile devices during recharge periods, when the CPU is typically underutilized[50], to perform biomolecular structure prediction tasks.

Similarly, the **Human Proteome Folding Project**[51] was a combined effort between two Rosetta labs, the *Institute for Systems Biology*, and IBM's *World Community Grid*. It was launched in 2004 with the goal to predict protein structures in the human genome, running for 9 years. Its successor, the *Microbiome Immunity Project*[52] was launched in 2017 with the goal to predict protein structures from the human gut microbiome.

Dissemination and interaction with users

Packaging and platforms

Rosetta simulations typically require High Performance Computing (HPC) resources, which are often public or university-owned. In response to the heterogeneity of available HPC resources, our software must be buildable across a wide range of hardware and compute environments. We customized the build system SCons[53] to allow users to specify system-specific compilation settings and select a particular build mode (e.g. a *Message Passing Interface* (MPI) mode for clusters that support or require it). Precompiled binary builds for 64-bit Linux and Mac distributions are available along with weekly source code releases (since 2013). Larger, numbered releases occur once or twice annually. The software suite runs on the three major operating systems (Linux, macOS, and Windows through the Windows Subsystem for Linux) and hardware architectures currently in use (x86-64 Intel/AMD chips, IBM Power architecture [e.g. Blue Gene supercomputers], and ARM for android-based cell phones and tablets [for Rosetta@home]). In the past, DEC Alpha, Cray, and Intel Itanium architectures were also supported.

Webservers

One path to perform simulations as described in published manuscripts is through webservers that present a limited number of parameters to the user. The Robetta server[54] was deployed in 2003 and is commonly used by scientists inside and outside of our community for fragment generation[55], domain parsing and structure prediction[56,57], alanine scanning[58], and DNA interface scanning[59]. Individual labs have also created servers for design[60], flexible backbone modeling and design[61], flexible peptide docking[62], and protein-protein interface identification[63]. In 2013, the ROSIE server framework[64] was created to simplify setting up new webservers for Rosetta applications. As of 2019, ROSIE hosts 21 modeling protocols, has >7,300 registered users and has run >75,000 jobs for a total of nearly 7.8 million CPU hours.

Licensing and commercialization

When developing software, it is important to think about licensing to answer the following questions: (1) by whom and under what circumstances will the software be used?; (2) how will contributions to the codebase be made, and from whom will they be accepted?; (3) will revenue be generated and, if so, how and what will it be spent on?; and, (4) will the choice of license restrict the use of external software libraries?

The Rosetta software is released under a unique licensing model, tied to its history[8]. Initially, when the software was developed solely by students and postdocs in David Baker's lab at the University of Washington, it was clear that sharing openly and avoiding disputes over ownership enabled innovative, rapid scientific progress. As these trainees started their own research groups at other institutions, they sought a licensing model to preserve the collaborative spirit and encourage sharing by removing barriers to inter-institutional development, leading to the establishment of the RosettaCommons. By joining the RosettaCommons, member institutions agree to provide non-exclusive rights to the copyright for contributions made to the codebase, and developers sign an agreement assigning copyright of code written by them to their institution. The software is distributed free of charge to academic, governmental, and other non-profit institutions. Commercial entities support continued development by paying a reasonable licensing fee that is based on the number of employees (more detail in [S5 Text](#)).

Revenue from commercial licenses, as well as government or private funding, is used to support meetings, software maintenance, outreach activities, trainings, lab exchanges and hackathons, mini-grants, and service awards to incentivize and reward community support. RosettaCommons PIs can apply for mini-grants, which are typically awarded to maintain, improve, extend and support the codebase, for instance for energy function improvements, rewrites of major frameworks, development of user interfaces and documentation, etc.

Member institutions of the RosettaCommons collectively hold the copyright to the source code but make no claim to the IP associated with products derived via use of the software (*e.g.* new protein therapeutics). Such IP is owned by the licensee, whether such licensee is academic or commercial. This has allowed spin-off companies to flourish that either use our software themselves under a commercial license to perform research and develop products/services, or such companies have licensed IP created by Rosetta academic licensees to allow product development. There are currently several commercial entities based entirely or in-part on our software, including the Rosetta Design Group, Arzeda, Cyrus Biotechnology, PVP Biologics, Neoleukin Therapeutics, and Lyell Immunopharma (see [S6 Text](#)).

Documentation

Rosetta is an inherently complex piece of software written for and by experts in the field, which can result in a complicated interface with lots of options. To ensure that scientists can use the software effectively, our developers have emphasized robust documentation with example workflows of real-world use cases called protocol captures[8,9]. Protocol captures force a developer to consider the perspective of end-users who may have different biological problems to solve, a differing scientific mindset, and different technical skills. Since scientifically focused grants (for instance from government agencies) typically do not provide time for code maintenance and writing documentation, the mini-grants, XRWs, and training or service awards help incentivize and recognize this important work.

Over the years, our community has used several types of documentation. To assist developers, the coding conventions for C++ (and later Python) include the use of comment-based API-level in-code documentation (**Doxygen**[65]). The combination of coding conventions with the requirement to write in-code documentation facilitates reading and understanding code written by someone else. We recently developed a Python-based code template system for frequently used classes to continue to improve both consistency and code commenting, while significantly reducing development and debugging time. While in-code Doxygen documentation is useful for developers, it is less beneficial to end-users because it describes internal functions and not the interface to an executable or RosettaScripts component.

Our user base spans thousands of users who require support, and our developers become users once they run someone else's code. This provides an incentive to write effective documentation[38], tutorials and protocol captures to answer common questions. Our documentation was transferred from an initial Doxygen-based[65] collection of application documentation pages to a **Gollum wiki**[38] and then greatly expanded upon during two *XRWs*. The Gollum wiki is split into user-facing and developer-facing documentation. The former contains information about released code, tutorials and generally helpful information for our users. The latter documentation contains information about experimental features pre-release and technical information. Both user-facing and developer-facing documentation are easily editable by RosettaCommons members and up-to-date with the current version of the software; as such, its detail, accuracy, and searchability have been transformative for both users and developers. This effort, combined with a push in the community for **protocol captures** in the supplementary material of published papers, have drastically improved quantity and quality of documentation. Another starting point for users is reviewing command lines used in the provided tests or demos for various applications. In addition to documentation, we have focused on improving error messages, for instance *via* XSD in RosettaScripts, to help users debug their custom scripts. Not all questions and use cases can be anticipated, so we also have a forum[39] to provide support and use the feedback from our collaborators to advance the software.

Remaining challenges

Although we have addressed many problems over the last 20 years, there are still several outstanding issues with respect to the three main areas explored in this paper: (1) Technical: The tension between best practices in software development and rapid scientific progress means we must continually provide incentives for prioritizing maintenance, usability, and reproducibility. Additionally, we are currently reconsidering the basic abstractions and data structures that we have relied on for over ten years to make use of massively parallel hardware architectures (e.g. GPUs). (2) Social: The enormous growth of our community contributes to questions of community and lab membership, the effectiveness of our diversity, equity, and inclusivity efforts, how to preserve the small, informal vibe as we continue to grow, and how to appropriately give credit (i.e. authorship) in an environment that is so clearly a team effort. (3) Dissemination: The diversity of applications, workflows and computer architectures make it challenging to ensure that outside users can easily integrate Rosetta into their workflows, and provide resources for backend compute for web servers, targeting a broader range of scientists. Fifteen specific challenges are detailed in [S7 Text](#).

Conclusions

Here we use the Rosetta software as an example to highlight the successes, mistakes made, and lessons learned over the past two decades as we developed a scientific software tool in a global, diverse, and distributed community. Our software was created mostly in an academic environment and is widely used beyond the laboratories in which it was created. We demonstrate the necessity of mastering the technical aspects of software development to create a package that allows the use of complex tools to drive the cutting edge in computational structural biology research.

Reasons for success in our community are numerous and largely intertwined. First and foremost, our community has lowered the barriers for scientific collaboration and thus enabled groundbreaking science and faster implementation of good ideas. On the technical side, we benefited from early established coding conventions, version control on GitHub, and

extensive code-testing through an integrated framework. We further maintain close interactions with users inside and outside of our community, who enable us to think outside of our established patterns. Understanding user needs and how our tools are actually used (instead of how we think that they are used) drives us to continuously improve our software for application to “real-world” scientific questions. To best take advantage of our broad developer community and to best support the users in our community, we created easily editable documentation that proved transformative in completing the coverage of the documentation and in supporting our large external user base.

Most importantly, scientific software cannot be sustained without a community around it, and we rely heavily on our community for training, communication, hackathons, and sharing expertise, whether scientific or related to software engineering. With the RosettaCommons, we have established an open community of developers that thrives on collaboration instead of competition: our common goal is to advance science. We are not afraid of tackling the most challenging scientific questions; in fact, we are motivated by them. Because our community is broad with a flat hierarchy, members have a voice irrespective of career level and are encouraged to contribute. Because of this, our community has established its own agreed-upon standards that members abide by. Forming such a tight-knit and collaborative community would not have been possible without a solid licensing model that was the result of early, thoughtful decisions that benefitted both our software and community over time. The agreement signed by all of the institutional members and developers of the RosettaCommons enables us to share and develop code independent of institution and commercial licensing of Rosetta allows industry to indirectly support maintenance and continued development through licensing fees.

Science is increasingly performed in teams to address complex questions with advanced methods and there is a need to address the challenges of achieving scientific goals in large groups[1]. The challenges we faced are remarkably similar to those noted by the National Academy’s Committee on Team Science[1]; we believe that these actions and lessons are independent of scientific discipline and therefore transferable to other communities. Our diverse community has thrived by committing to our shared goals and strong belief that, in fact, we are better together.

Supporting information

S1 Fig. Growth in Rosetta PIs and conference attendees.

(DOCX)

S1 Text. Specific tests for Rosetta running on our automated testing framework.

(DOCX)

S2 Text. Typical development workflow in the Rosetta community.

(DOCX)

S3 Text. Descriptions of the Rosetta board positions.

(DOCX)

S4 Text. Values Statement: Equality and Inclusion in the RosettaCommons.

(DOCX)

S5 Text. More detail on specific licenses.

(DOCX)

S6 Text. Commercial spin-off companies from the Rosetta community.

(DOCX)

S7 Text. Details of Remaining Collective Challenges.
(DOCX)

S1 List. Members of the RosettaCommons Consortium.
(DOCX)

Acknowledgments

We thank Pat Gunn and Erin Dolan for comments on the manuscript.

References

1. Council NR. Enhancing the Effectiveness of Team Science. Washington, D.C.: National Academies Press; 2015. <https://doi.org/10.17226/19007> PMID: 26247083
2. Wiltshire TJ, Butner JE, Fiore SM. Problem-Solving Phase Transitions During Team Collaboration. *Cogn Sci*. 2018; 42: 129–167. <https://doi.org/10.1111/cogs.12482> PMID: 28213928
3. RosettaCommons. [cited 2 Jul 2019]. Available: <https://www.rosettacommons.org/>
4. Koehler Leman J, Weitzner BD, Lewis SM, Consortium R, Bonneau R. Macromolecular Modeling and Design in Rosetta: New Methods and Frameworks. preprints.org. 2019 [cited 13 May 2019]. <https://doi.org/10.20944/PREPRINTS201904.0263.V1>
5. Alford RF, Leaver-Fay A, Jeliazkov JR, O'Meara MJ, Dimaio FP, Park H, et al. The Rosetta all-atom energy function for macromolecular modeling and design. *J Chem Theory Comput*. 2017; 13: 1–35. <https://doi.org/10.1021/acs.jctc.6b01111> PMID: 28068772
6. Hannay JE, MacLeod C, Singer J, Langtangen HP, Pfahl D, Wilson G. How do scientists develop and use scientific software? 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering. IEEE; 2009. pp. 1–8. <https://doi.org/10.1109/SECSE.2009.5069155>
7. Nowogrodzki A. How to support open-source software and stay sane. *Nature*. 2019; 571: 133–134. <https://doi.org/10.1038/d41586-019-02046-0> PMID: 31263262
8. Renfrew PD, Campbell G, Strauss CEM, Bonneau R. The 2010 Rosetta Developers Meeting: Macromolecular Prediction and Design Meets Reproducible Publishing. Uversky VN, editor. *PLoS One*. 2011; 6: e22431. <https://doi.org/10.1371/journal.pone.0022431> PMID: 21909349
9. Bender BJ, Cisneros A, Duran AM, Finn JA, Fu D, Lokits AD, et al. Protocols for Molecular Modeling with Rosetta3 and RosettaScripts. *Biochemistry*. 2016; acs.biochem.6b00444. <https://doi.org/10.1021/acs.biochem.6b00444> PMID: 27490953
10. Allison DB, Shiffrin RM, Stodden V. Reproducibility of research: Issues and proposed remedies. *Proc Natl Acad Sci U S A*. 2018; 115: 2561–2562. <https://doi.org/10.1073/pnas.1802324115> PMID: 29531033
11. Stodden V, Mcnutt M, Bailey DH, Deelman E, Gil Y, Hanson B, et al. Enhancing reproducibility for computational methods. *Science (80-)*. 2016; 354: 1240–41. <https://doi.org/10.1126/science.aah6168> PMID: 27940837
12. Leaver-Fay A, Tyka M, Lewis SM, Lange OF, Thompson JM, Jacak R, et al. ROSETTA3: An Object-Oriented Software Suite for the Simulation and Design of Macromolecules. *Methods Enzymol*. 2011; 487: 545–74. <https://doi.org/10.1016/B978-0-12-381270-4.00019-6> PMID: 21187238
13. The Noun Project. Available: <https://thenounproject.com/>
14. Simons KT, Kooperberg C, Huang E, Baker D. Assembly of Protein Tertiary Structures from Fragments with Similar Local Sequences using Simulated Annealing and Bayesian Scoring Functions. 1997.
15. Simons KT, Ruczinski I, Kooperberg C, Fox BA, Bystroff C, Baker D. Improved recognition of native-like protein structures using a combination of sequence-dependent and sequence-independent features of proteins. *Proteins*. 1999; 34: 82–95. Available: <http://www.ncbi.nlm.nih.gov/pubmed/10336385> [https://doi.org/10.1002/\(sici\)1097-0134\(19990101\)34:1<82::aid-prot7>3.0.co;2-a](https://doi.org/10.1002/(sici)1097-0134(19990101)34:1<82::aid-prot7>3.0.co;2-a) PMID: 10336385
16. Simons KT, Bonneau R, Ruczinski I, Baker D. Ab initio protein structure prediction of CASP III targets using ROSETTA. *Proteins*. 1999; Suppl 3: 171–6. Available: <http://www.ncbi.nlm.nih.gov/pubmed/10526365>
17. Bonneau R, Tsai J, Ruczinski I, Chivian D, Rohl C, Strauss CEM, et al. Rosetta in CASP4: Progress in Ab Initio Protein Structure Prediction. 2002; 126: 119–126. <https://doi.org/10.1002/prot.1170> PMID: 11835488

18. Fleishman SJ, Leaver-Fay A, Corn JE, Strauch E-MM, Khare SD, Koga N, et al. RosettaScripts: A scripting language interface to the Rosetta Macromolecular modeling suite. *PLoS One*. 2011; 6: 1–10. <https://doi.org/10.1371/journal.pone.0020161> PMID: 21731610
19. Chaudhury S, Lyskov S, Gray JJ. PyRosetta: a script-based interface for implementing molecular modeling algorithms using Rosetta. *Bioinformatics*. 2010; 26: 689–691. <https://doi.org/10.1093/bioinformatics/btq007> PMID: 20061306
20. Gray, J. J., Chaudhury, S., Lyskov, S., and Labonte JW. The PyRosetta Interactive Platform for Protein Structure Prediction and Design: A Set of Educational Modules. 2014 [cited 20 Oct 2015]. Available: <http://www.amazon.com/PyRosetta-Interactive-Platform-Structure-Prediction/dp/1500968277>
21. CVS—Concurrent Versions System. In: <http://savannah.nongnu.org/projects/cvs>.
22. SVN—Apache Subversion. In: <https://subversion.apache.org/>.
23. GitHub. In: <https://github.com/>.
24. Buildbot. In: <https://buildbot.net/>.
25. Jenkins. In: <https://jenkins.io/>.
26. GitLab. In: <https://about.gitlab.com/>.
27. Travis CI—continuous integration. In: <https://travis-ci.org/>.
28. Rosetta Coding Conventions. [cited 3 Jul 2019]. Available: https://www.rosettacommons.org/docs/latest/development_documentation/Coding-Conventions
29. Cooper S, Khatib F, Treuille A, Barbero J, Lee J, Beenen M, et al. Predicting protein structures with a multiplayer online game. *Nature*. 2010; 466: 756–760. <https://doi.org/10.1038/nature09304> PMID: 20686574
30. Kleffner R, Flatten J, Leaver-Fay A, Baker D, Siegel JB, Khatib F, et al. Foldit Standalone: a video game-derived protein structure manipulation interface using Rosetta. Valencia A, editor. *Bioinformatics*. 2017; 33: 2765–2767. <https://doi.org/10.1093/bioinformatics/btx283> PMID: 28481970
31. Khatib F, Cooper S, Tyka MD, Xu K, Makedon I, Popovic Z, et al. Algorithm discovery by protein folding game players. *Proc Natl Acad Sci U S A*. 2011; 108: 18949–53. <https://doi.org/10.1073/pnas.1115898108> PMID: 22065763
32. Dsilva L, Mittal S, Koepnick B, Flatten J, Cooper S, Horowitz S. Creating custom Foldit puzzles for teaching biochemistry. *Biochem Mol Biol Educ*. 2019; 47: 133–139. <https://doi.org/10.1002/bmb.21208> PMID: 30638297
33. Rosetta user documentation. In: <https://www.rosettacommons.org/docs/latest/Home> [Internet]. Available: <https://www.rosettacommons.org/docs/latest/Home>
34. Nerli S, Sgourakis NG. CS-ROSETTA. *Methods Enzymol*. 2018 [cited 24 Nov 2018]. <https://doi.org/10.1016/BS.MIE.2018.07.005> PMID: 30611429
35. Nerli S, McShan AC, Sgourakis NG. Chemical shift-based methods in NMR structure determination. *Prog Nucl Magn Reson Spectrosc*. 2018;106–107: 1–25. <https://doi.org/10.1016/J.PNMRS.2018.03.002> PMID: 31047599
36. Schenkelberg CD, Bystroff C. InteractiveROSETTA: A graphical user interface for the PyRosetta protein modeling suite. *Bioinformatics*. 2015 [cited 2 Sep 2015]. <https://doi.org/10.1093/bioinformatics/btv492> PMID: 26315900
37. MediaWiki. In: <https://www.mediawiki.org/wiki/MediaWiki>.
38. Gollum wiki. In: <https://github.com/gollum/gollum>.
39. RosettaCommons Forum. [cited 7 Jul 2019]. Available: <https://www.rosettacommons.org/forum>
40. Rosetta workshops. In: <http://structbio.vanderbilt.edu/comp/workshops/rosetta/> [Internet]. Available: <http://structbio.vanderbilt.edu/comp/workshops/rosetta/>
41. RosettaCommons YouTube channel. In: <https://www.youtube.com/channel/UC5urBzvutWGrJyLvpQ4AFpw>.
42. RosettaCommons Code of Conduct. [cited 2 Jul 2019]. Available: <https://www.rosettacommons.org/codeofconduct>
43. RosettaCommons Diversity Statement. [cited 2 Jul 2019]. Available: <https://www.rosettacommons.org/diversity>
44. Thompson M, Sekaquaptewa D. When Being Different Is Detrimental: Solo Status and the Performance of Women and Racial Minorities. *Anal Soc Issues Public Policy*. 2002; 2: 183–203. <https://doi.org/10.1111/j.1530-2415.2002.00037.x>
45. Rosetta Commons Research Experience for Undergraduates | RosettaCommons. [cited 7 Jul 2019]. Available: <https://www.rosettacommons.org/about/intern>

46. Alford RF, Leaver-Fay A, Gonzales L, Dolan EL, Gray JJ. A cyber-linked undergraduate research experience in computational biomolecular structure prediction and design. Ouellette F, editor. *PLOS Comput Biol*. 2017; 13: e1005837. <https://doi.org/10.1371/journal.pcbi.1005837> PMID: 29216185
47. Das R, Qian B, Raman S, Vernon R, Thompson J, Bradley P, et al. Structure prediction for CASP7 targets using extensive all-atom refinement with Rosetta@home. *Proteins Struct Funct Bioinforma*. 2007; 69: 118–128. <https://doi.org/10.1002/prot.21636> PMID: 17894356
48. Rosetta@home. In: <https://boinc.bakerlab.org/>.
49. BOINC—Berkeley Open Infrastructure for Network Computing. In: <http://boinc.berkeley.edu/>.
50. Hosseinzadeh P, Bhardwaj G, Mulligan VK, Shortridge MD, Craven TW, Pardo-Avila F, et al. Comprehensive computational design of ordered peptide macrocycles. *Science (80-)*. 2017; 358: 1461–1466. <https://doi.org/10.1126/science.aap7577> PMID: 29242347
51. Drew K, Winters P, Butterfoss GL, Berstis V, Uplinger K, Armstrong J, et al. The Proteome Folding Project: proteome-scale prediction of structure and function. *Genome Res*. 2011; 21: 1981–94. <https://doi.org/10.1101/gr.121475.111> PMID: 21824995
52. Microbiome Immunity Project on the World Community Grid. In: <https://www.worldcommunitygrid.org/research/mip1/overview.do>.
53. SCons: A software construction tool—SCons. [cited 8 Jul 2019]. Available: <https://scons.org/>
54. Kim DE, Chivian D, Baker D. Protein structure prediction and analysis using the Robetta server. 2004; 32: 526–531. <https://doi.org/10.1093/nar/gkh468> PMID: 15215442
55. Gront D, Kulp DW, Vernon RM, Strauss CEM, Baker D. Generalized Fragment Picking in Rosetta: Design, Protocols and Applications. 2011; 6: e23294. <https://doi.org/10.1371/journal.pone.0023294> PMID: 21887241
56. Song Y, Dimaio F, Wang RY-RR, Kim DE, Miles C, Brunette T, et al. High-resolution comparative modeling with RosettaCM. *Structure*. 2013; 21: 1735–1742. <https://doi.org/10.1016/j.str.2013.08.005> PMID: 24035711
57. Raman S, Vernon R, Thompson J, Tyka M, Sadreyev R, Pei J, et al. Structure prediction for CASP8 with all-atom refinement using Rosetta. *Proteins*. 2009; 77 Suppl 9: 89–99. <https://doi.org/10.1002/prot.22540> PMID: 19701941
58. Kortemme T, Kim DE, Baker D. Computational alanine scanning of protein-protein interfaces. *Sci STKE*. 2004; 2004: pl2. <https://doi.org/10.1126/stke.2192004pl2> PMID: 14872095
59. Ashworth J, Baker D. Assessment of the optimization of affinity and specificity at protein-DNA interfaces. *Nucleic Acids Res*. 2009; 37: e73. <https://doi.org/10.1093/nar/gkp242> PMID: 19389725
60. Rosetta Design server. In: <http://rosettadesign.med.unc.edu/>.
61. Rosetta Backrub server. In: <https://kortemmeweb.ucsf.edu/backrub/cgi-bin/rosettaweb.py?query=index>.
62. FlexPepDock server. In: <http://flexpepdock.furmanlab.cs.huji.ac.il/>.
63. FunHunt server. In: <http://funhunt.furmanlab.cs.huji.ac.il/>.
64. Lyskov S, Chou FC, Conch??ir S??, Der BS, Drew K, Kuroda D, et al. Serverification of Molecular Modeling Applications: The Rosetta Online Server That Includes Everyone (ROSIE). *PLoS One*. 2013; 8: 5–7. <https://doi.org/10.1371/journal.pone.0063906> PMID: 23717507
65. Doxygen. In: <http://www.doxygen.org/>.